Batch G. D.

Downside can't work on large dataset

epoch = 10

for i in range(10):

y_hat = np.dot(x, w) + b

↳ vectorization (Instead using loop ∴ use vector multiplication)

✓ 50 values

y hat, y → loss.

↳ w, b update

$w_n = w_o - \eta \frac{\partial L}{\partial w}$

→ loss.

∴ Number of update = No. of epoch.

Stochastic Gradient descent

epoch → 10

for in range(10): → shuffle

for i in range (x.shape[0])

↳ one random point

↳ y_hat → forward

↳ loss

↳ w, b update

→ $w_n = w_0 - \eta \frac{\partial L}{\partial w}$

↓

avg loss epoch.

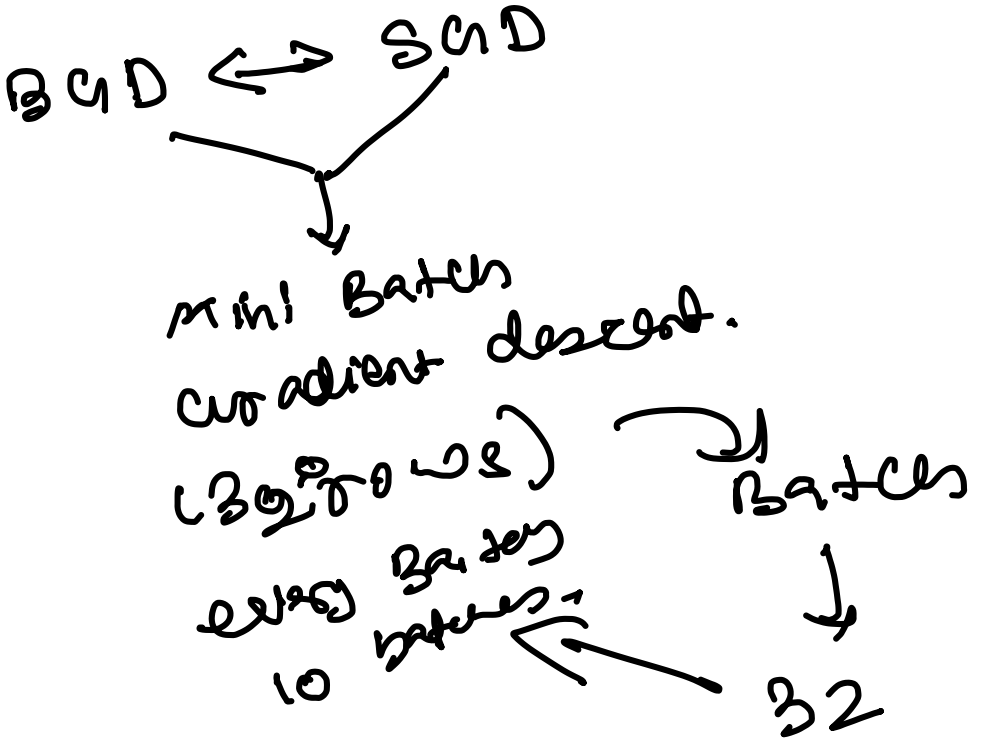
∴ weight updated
2 10 × 50 = 500 (frequency of weight is higher)

Batch	SGD.
→ less frequency	more frequency.
→ 1 epoch (1 update)	epochs not rows (updates)
→ Batch size = shape of rows.	Batch size = 1
→ SGD as the number of rows.	to converge (give same result) of update

which is the faster

In code if we give Batch size parameter 1 then it is stochastic gradient descent and given size, shape then it is Batch gradient descent.

Mini Batch Gradient Descent



for i in epoch
  for j in number of
  Batches
    ↳ y-pred (vectors
    ↳ loss
    ↳ update

Why batch-size is power of multiple (2).
↳ 2, 4, 8, ... ↳ Gpu effective,

Ram-Obimsah

→ what if batch-size does not divide # rows properly?

# of rows $n = 400$

batch-size = 150

# of batches $\frac{400}{150} = 2.66$

150, 150, 100